Università
della
Svizzera
italiana

Image and Video Processing **2023**

Student: Filippo Casari, Alessandro De Grandi

Report for Assignment 1

1. Point Operations

1.1. Tone mapping & Linearization

First, we used the function *imread* to load the image, and then we applied 3 transformations such as inverting gamma function, linear multiplications (by multiplying by 2, more brightness) and finally contrast enhancement (power of 1.8).

Figure 1: First Exercise

1.2. Color correction

1.2.1. Pixel-based correction

To perform pixel-based correction we selected a pixel that should be gray in the initial image, such as a pixel on the laptop's frame. The goal is to find gain values for each channel such that this pixel actually becomes gray in the color corrected image. To achieve this we calculated the average value over the whole image (0.4953 which is gray), and divided it by the selected pixel values for each

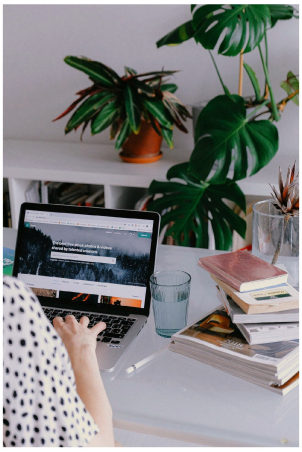
channel to obtain the per channel gains. Then multiplied the gains to the initial image channels to obtain a color corrected image (fig. 2b).

1.2.2. Gray-world assumption

Similarly to the previous method, but instead of using the channels values of a pixel we used the average values over the channels. So in this case the goal is to make the average value gray for each channel of the color corrected image. So by dividing the average value over the whole image (or just using the color gray 0.5) by the average over each channel we obtain per channel gains. Then we multiplied the gains to the initial image channels to obtain a color corrected image (fig.2c).

(a) Input

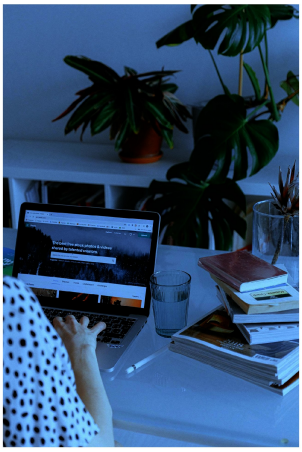
(b) Pixel-Based correction



(c) Gray-world correction

1.2.3. With inverse gamma correction

When applying linear operations it would be more correct to first perform inverse gamma correction, because otherwise we are using linear operations on non-linear images where intensities have been mapped using an exponential function. Since light transport is linear, using linear color for image processing is important to ensure that the mathematical operations applied on the image are in accordance with how pictures are acquired. This time we performed inverse gamma correction on the input image, and then re-applied gamma correction after color correction. The results are visually similar, except the Pixel-Based correction which appear brighter.



(a) Input inv-gamma-corrected

(b) Pixel-Based correction

(c) Gray-world correction

1.3. Histograms

In order to draw the histogram for the ferrari image, we wrote a function called "*compute_histogram*" which returns the histogram (fig. 4) for the channel chosen and the corresponding density function (fig. 5). As one can see, the values are not distributed uniformly.

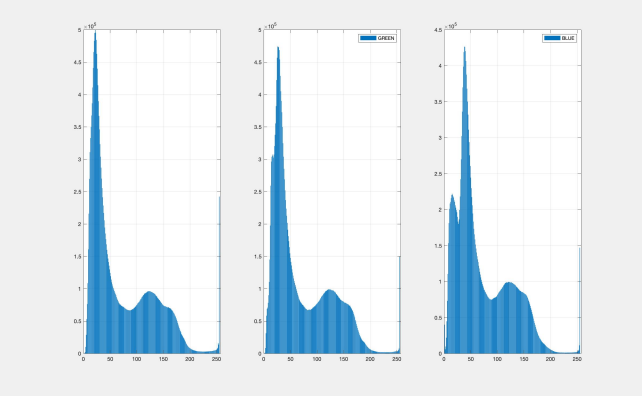


Figure 4: Histogram for each channel

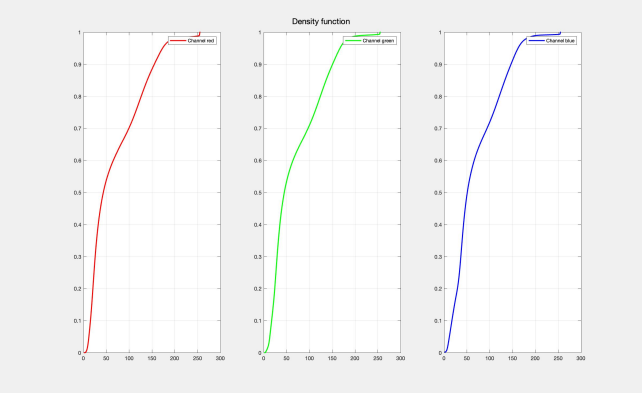


Figure 5: Density for each channel

1.3.1. Global Equalization

For this exercise we wrote the function *global_equalizer* which returns: the new image channel equalized and the corresponding new image histogram. I show below the new image (concatenating all color channels), fig, 6.

Figure 6: Global Equalization

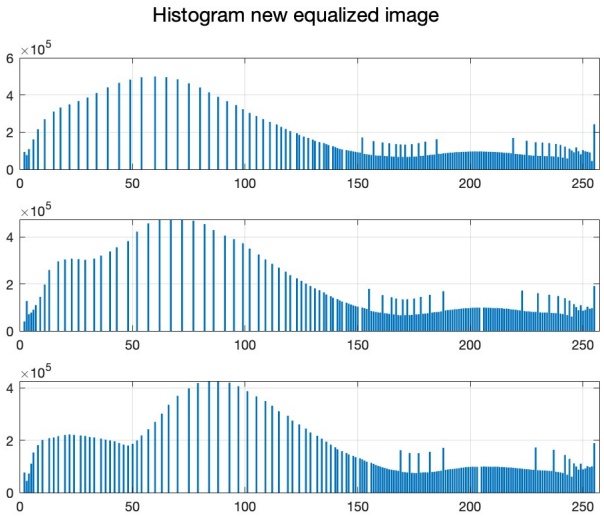


Figure 7: Histogram (for each channel) after Global Equalization

1.3.2. Local Equalization

For this purpose, we wrote a function called *local_equalizer* which takes as input the image, the index corresponding to the specified channel, and the window size. The default window size is 1000, but can be replaced with other values. This function exploits the previous function global equalizer locally passing through the entire image. As one can see this method leads to the artifact shown below (fig. 8).

Figure 8: Local Equalization

1.3.3. Adaptive Equalization

The method which implements this functionality is called *"adaptive_equalizer"*. This is the slowest method because it has to iterate over each pixel, but the result is much better than the previous one.

Figure 9: Adaptive Equalization

Note that the image is low-quality (fig. 9) because we resized the input for computational reason. Otherwise it would have taken longer time to iterate throughout the entire image (considering that the operation is performed for each pixel).

1.4. Manual histogram equalization

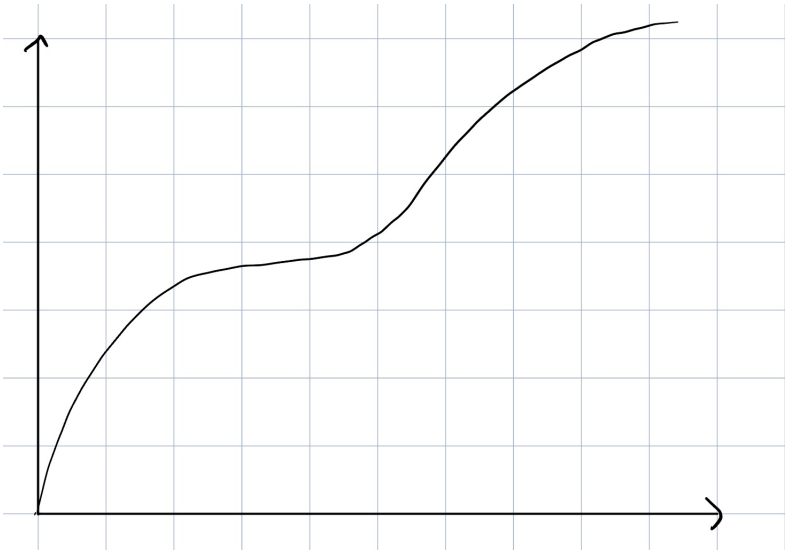


Figure 10: pixel transformation

1.5. Thresholding & matting

To separate the subject of the image from the background we first looked at its histogram 16, to try and manually pick threshold values. Assuming that the most frequent values, that can be seen as a peak in the histogram, represent the background. And the cat is mostly black and white, we selected two thresholds per channel around the peak, so all the pixels values in between could be replaced with the new background. The results of this method were not great, and adjusting manually the threshold values was a tedious process. We also tried the same method with a greyscale version of the cat image that should be easier to binarize to obtain the mask, but with worst results.

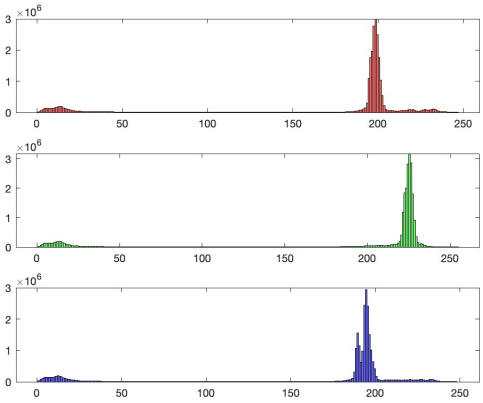


Figure 11: Cat color histogram

So instead we tried picking n values from the background with the ginput function, and using the minimum and maximum values per channel as thresholds, and after some tries we got better results, and we obtained a decent α-mask of the cat image. That can be applied in the following formula to replace any image to the background:

$$f_{i,j} = (1 - \alpha_{i,j}) \cdot f1_{i,j} + \alpha_{i,j} \cdot f2_{i,j}$$

(a) Cat mask

(b) Cat mask inverted

(a) Cat & Ferrari

(b) Cat & Nyancats

1.6. How many bits is enough

$$\frac{I_{max}}{I_{min}} = (1+k)^{N-1}$$
$$I_{min} = 0.5\ cd/m^2$$
$$I_{max} = 1000\ cd/m^2$$
$$k = 0.01$$
$$N = \log_{1+0.01}\left(\frac{1000}{0.5}\right) + 1 = 765\ levels\ of\ intensity\ that\ can\ be\ encoded\ with\ 10\ bits$$

2. Bonus: use your own pictures

(a) Tony

(b) Toni with threshold/cropped/padded

(a) Taimoor

(b) hippie Taimoor

Figure 16: Tony & Taimoor